HetSC User Manual

ANNEXE A:
HetSC Library

January, 2008

www.teisa.unican.es/HetSC

GIM Group

TEISA Dpt.

University of Cantabria

Authors: F. Herrera & E.Villar

The maintenance of this user manual and of the last version of the HetSC library has been funded by the ANSDRES IST project (http://andres.offis.de).

Document Version:

Version	Date	Content
1.0	August , 2006	The initial version of HetSC User Manual presented as a quick user guide
1.1	July, 2007	HetSC User Manual. Added Graphical Representation, General Rules and KPN and CSP specification rules. Added HetSC library elements and specification guidelines, checks and reports.
1.2	January, 2008	Adaptation to HetSCv1.2 release and implementation of comments from UC internal revision (Margarita Díez).

Index:

1 ANNEXE A: HetSC Library

1.1 Introduction

The HetSC methodology is based on a set of specification facilities. Whenever possible they are taken from the OSCI SystemC library. The *HetSC library*, a proof-of-concept library associated to the HetSC methodology, provides a set of facilities to cover the deficiencies of the SystemC core language for heterogeneous specification. Therefore, in order to enable the compilation and execution of HetSC specifications, the HetSC user only needs to install the OSIC SystemC library and the HetSC library.

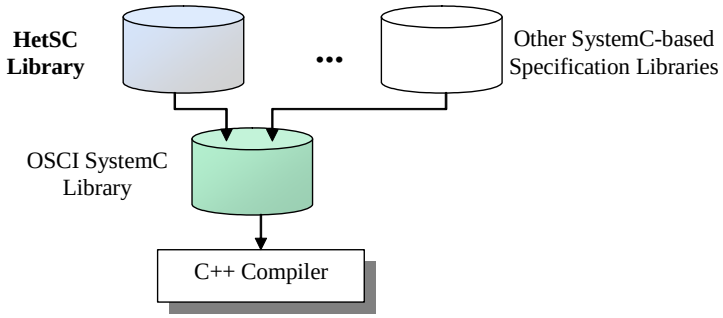

Figure 1. The HetSC library is installed over the OSCI SystemC library.

As mentioned, the support of some specific MoCs requires new facilities, not included by the SystemC library. These facilities are:

- Specification facilities, such as interfaces, channels, etc. They provide the specific semantic content and abstraction level required by their corresponding MoC.
- Facilities for specifying HW/SW partition and other information which is used by other system-level design activities (such as software generation or time profiling).
- MoC rule checkers. These are facilities, often transparent for the HetSC user, which help to detect and locate MoC rule violations in the specification.
- Facilities for generating MoC specific reports.
- Facilities for assisting the debugging task of concurrent specifications.

1.2 Development Platform

The HetSC library is installed over the SystemC core library. As it is known, it is necessary a C++ compiler to install the SystemC core library. The last version available is the **HetSCv1.2** library. This version has been checked on platform combinations with the following characteristics:

- OS:
  - GNU/Linux kernel 2.6.3 (32 bits). Distribution: Mandrake 10.0.
  - GNU/Linux kernel 2.6.23 (64 bits). Distribution: Fedora 8.
- COMPILER:
  - gcc-4.1.0 and gcc-4.1.2 compiler.
- SystemC-2.2.0.

HetSCv1.2 library does not keep backward compatibility with previous releases of the SystemC library (SystemC 2.1, 2.0 etc). The use of HetSCv1.2 library is recommended since SystemC-2.2.0 keeps compatibility with the IEEE1666 standard **¡Error! No se encuentra el origen de la referencia.**. For previous versions of SystemC, use the previous versions of HetSC.

Minor modifications should be necessary to successfully install the HetSC v1.2 library on platforms based either on Unix or Windows/Cygwin.

1.3 TLM compatibility

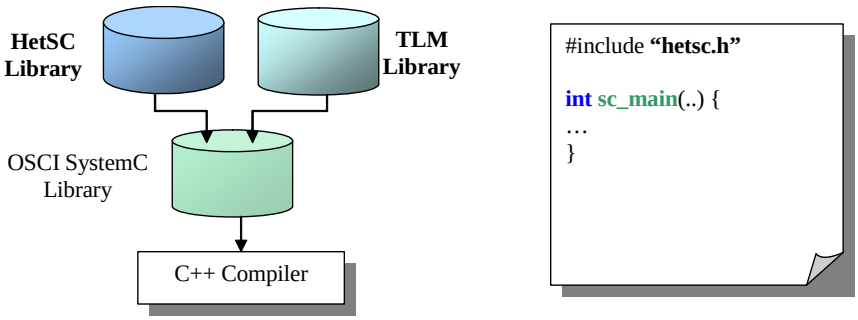

Figure 2. HetSC library facilities can support TLM interfaces.

Several channels of the HetSC library present TLM compatible interfaces. For instance, if desired, the *uc_fifo* channel template can be written either by means of the write(...) method or through the put(...) TLM standard method. Table 2 informs which channels currently support TLM standard interfaces. For those cases, this documentation explains which standard TLM interfaces&methods are used.

For installing the library with this compatibility enabled, ensure that the _TLM_COMPATIBILITY variable is defined in the *compatibility.h* file (in $(HETSC_PKGDIR)/src/common) before the installation of the library.

To compile the examples for checking TLM compatibility features, ensure that you have the TLM library installed (TLM 2.0) and edit the *Makefile.defs* example (in

the $(HETSC_PKGDIR)/examples directory) to make the TLM_PATH variable to point where TLM library is installed in your system.

1.4 Compatibility with SystemC-AMS

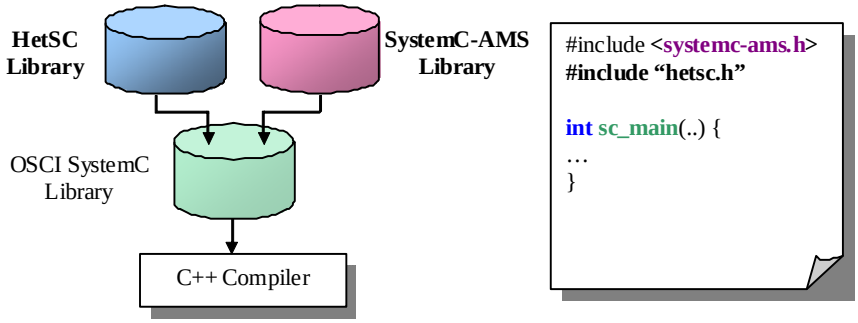

Figure 3. The HetSC library can cooperate with the SystemC-AMS.

The order of installation of HetSC and SystemC-AMS libraries is irrelevant. In order to use the border channels included for direct connection between HetSC and SystemC-AMS, include first SystemC-AMS library (*"systemc-ams.h"*) and then the HetSC library (*"hetsc.h"*) in your HetSC/SystemC-AMS source code.

1.5 Installation

This section gives instructions to install the HetSCv1.2 library. For installing previous versions of the HetSC library, specific instructions are given in section **¡Error! No se encuentra el origen de la referencia.** and in the INSTALL file of the distributed sources.

In order to build and install the HetSCv1.2 library, follow these instructions:

1. Unzip and untar the *HetSCv1.2.tar.gz* file. A folder is created in the *untar_path* path. For instance, if the *HetSCv1.2.tar.gz* file is untared in */home/usr/src*, then the */home/usr/src/HetSC* directory is created (*untar_path=/home/usr/src/HetSC*).
2. Make sure that your compiler is a suitable version (the "*gcc –v*" command provides the compiler version). Make sure that the SystemC installation was done with the same *gcc* compiler version which will be used to install the HetSC library.
3. Set and export the SYSTEMC_PATH variable. For instance, in a Linux environment you write:

   $SYSTEMC_PATH=/home/user/soft/systemc-2.2.0/installdir

   $export SYSTEMC_PATH
4. You can either, set any of the following two variables:

- *INSTALLDIR:* This is the path where HetSC library will be installed. For instance, *INSTALLDIR = /home/usr/soft/hetsc*.

  - Note: An installation directory different from HETSC_PCKDIR is recommended, however it is not mandatory.

- *BUILDIR:* This is the path where the HetSC library is compiled. For instance, *BUILDIR=/home/usr/tmp/hetsc/buildir*.

Or you can leave them in their default values. As default value, *INSTALLDIR* takes the current directory, while *BUILDIR* takes the value given by the next expression: BUILDIR=$(INSTALLDIR)/buildir.

- Move to the *$HETSC_PKGDIR* directory. If you are installing the HetSC library from another directory set the *HETSC_PKGDIR* variable with the *untar_path* value. For instance, taking into account the unzip&untar directory in the first point, *HETSC_PKGDIR=/home/usr/src/HetSC*. When you move there, *HETSC_PKGDIR* variable takes by default this value.

5. Compile an install the library through the make command:

   $make

   These two steps can be separately done by executing:

   $make build (library compilation)

   $make install (library installation)

When the compilation&installation finishes, you should have the next installation tree installed:

$(INSTALLDIR)/

/include

/lib

/doc

$(BUILDIR)\

The $(INSTALLDIR) contains all the files needed for using the HetSC methodology. The */include* directory has all the HetSC source header files to be included in the sources of the HetSC specification, through the "*hetsc.h*" (or the "*general.h*") header file. The */lib* directory contains the *hetsc.a* static library which has to be linked to the specification object files to produce the executable specification. The */doc* contains HetSC documentation.

The $BUILDIR directory contains all the object files for compiling the HetSC library and can be removed after the installation.

1.6 Removing the build directory

You can remove object files to save disk space without affecting the installation. In order to do this, just execute the next command at the *$(HETSC_PKGDIR)* directory:

$make clean

1.7 Uninstall the HetSC library

The HetSC library can be uninstalled just by executing the next command at the *$(HETSC_PKGDIR)* directory:

$make uninstall

1.8 SystemC executable with HetSC

In order to use the HetSC library facilities in your SystemC code, you should include the *"general.h"* header (or the *"hetsc.h"* if you are not using other UC libraries) in your specification. This header already includes the *"systemc.h"* header. Before compiling your application, ensure that you include *$(INSTALLDIR)\include* path in the include paths, the static library path, *$(INSTALLDIR)\lib*, and the HetSC library it self.

For instance, if you write a specification file called *myspec.cpp*, then, in order to obtain the executable binary file *myspec.x* you can use the command:

$g++ -I$(INSTALLDIR)\include -L$(INSTALLDIR)\lib -o myspec.x -lhetsc myspec.cpp

From its v1.2 release, HetSC can be included in different compilation units. As a compilation example, lets assume you have two HetSC source files which are separately compiled, *system.cpp* and *testbech.cpp*, both including HetSC library. Then, you can apply the next sequence of commands.

$g++ -I$(INSTALLDIR)\include –c system.cpp

$g++ -I$(INSTALLDIR)\include –c testbench.cpp

$g++ -L$(INSTALLDIR)\lib -o myspec.x system.o testbench.o –lhetsc

1.9 Examples

The library comes with a set of simple examples which show you some of the features of the HetSC library and can be taken as templates for constructing your own examples.

To compile them, move to the */examples* directory in the source directory and edit the *Makefile.defs* file to indicate the installation directory. After that, type the next command:

$make –f Makefile.sys

Then all provided examples will be compiled. They can be cleaned (removal of object and executable files) through the next command.

$make –f Makefile.sys clean

If you want to compile a specific group of examples, then enter the specific directory and compile them with the same command. For example, if you want to compile only KPN examples, enter the */examples/KPN* directory and type:

$make –f Makefile.sys

Again, those examples can be cleaned with the clean command. You can also compile and clean single examples entering their specific directory and applying a rule of the Makefile.sys file. For instance, to compile the first example of KPN directory type:

$make –f Makefile.sys example1

Many of the examples are ready to, through some simple comment/uncomment of define clauses, to check many different features and behaviour of the HetSC library specification facilities. For it, you will need to recompile the specific example where you are making such editions.

1.10 Specification facilities

This section details enumerates the specification facilities provided by the HetSC library for heterogeneous specification.

1.10.1 HetSC Interfaces

The next table shows the interfaces currently provided by the HetSCv1.2 library:

MoC	Interfaces
CSP	**uc_caller_if<T>** **uc_accepter_if<T>** **uc_rv_if<T>** **uc_sync_if<T>**
PN/KPN	**sc_fifo interfaces ...**
PN(1) (with fifo of sizes=1) **Equivalent to HSDF (HSDF)**	**uc_simple_read_if<T>** **uc_simple_write_if<T>**
SDF	**uc_arc_introspection_if<T>** **uc_arc_prod_seq_if<T>** **uc_arc_prod_dir_if<T>** **uc_arc_cons_seq_if<T>** **uc_arc_cons_dir_if<T>** **uc_arc_prod_if<T>** **uc_arc_cons_if<T>** **uc_arc_if<T>**
SR	***uc_SR*_read_if<T>** ***uc_SR*_write_if<T>** ***uc_SR*_if<T>**
KPN/ **PN/** **CSP-REFINED**	**uc_sig_client_in_if<T>** **uc_sig_client_out_if<T>** **uc_sig_server_in_if<T>** **uc_sig_server_out_if<T>** **uc_sigclocked_client_in_if<T>** **uc_sigclocked_client_out_if<T>**

	uc_sigclocked_server_in_if<T> uc_sigclocked_server_out_if<T>
I/O	uc_uart_if

Table 1. HetSC interfaces.

1.10.2 HetSC channels

The next table shows the channels currently provided by the HetSCv1.2 library.

MoC	MoC channels	TLM interfaces
CSP	uc_frv	-
	uc_rv	yes
	uc_rv_uni	yes
	uc_rv_sync	-
PN	uc_fifo	yes
KPN	uc_inf_fifo	yes
HSDF	uc_simple_channel	-
SDF	uc_arc_seq	-
SR	*uc_SR*	-
PN-refined	uc_sigclocked_fifo	-
Experimental	uc_bucket uc_protected uc_shared	-

Table 2. HetSC channels.

MoC connection	Border Channels
PN-CSP	uc_fifo2rv
PN-SR	uc_fifo_SR
	*uc_SR*_fifo
KPN-SR	uc_inf_fifo_SR
	*uc_SR*_inf_fifo
RT(HW)-KPN	uc_signal_inf_fifo

	uc_inf_fifo_signal

Table 3. HetSC border channels.

In some cases, these channels do not enable data transfer. In those cases, HetSC channels and interfaces their respective interfaces are classes. In many other cases, HetSC channels and their respective interfaces enable some kind of data transfer. In those cases they are usually class templates. In some cases, these HetSC channels are unidirectional, that is, the enable data transfer in a single sense (from one process to the other). In these cases, the HetSC channel classes present a single data type parameter T, representing the type of data transferred by the channel. Some HetSC channels will be bidirectional, that is, they can transfer tokens between two processes in both directions. In the most general case, two template parameters will appear, reflecting that different data types can be transferred in each transfer sense.

In any case, once a channel instance has been declared, the data(s) type(s) transferred for the channel instance is(are) fixed during all the simulation.

Specific details will be given for each channel, in its context of use under a specific MoC in the following sections.

1.10.3 Utility for Concurrent Debugging

The HetSC user can use a macro function called *CH_MONITOR* to assist the debugging task of concurrent specifications in HetSC. To use it, the *CH_MONITOR* macro has to wrap every channel access to be monitored. It can wrap either an access to an inner channel (a channel declared within the ambit of a module) or a port access.

Then if, in the specification, a macro variable called *SC_DEBUG* is defined and a PERF macro variable is not defined, then, a report of the last blocking state of every process at the end of simulation is obtained. This is useful for a system-level debug of deadlock conditions present in the system-level specification.

The SC_DEBUG preprocesor variable can be defined during the installation of HetSC library (defining it in the *global_opts.h* header file used for HetSC library configuration). By default, SC_DEBUG is not defined in this file. Then, to use these debug facility, the user can define the SC_DEBUG variable before the inclusion of the "hetsc.h" header file.

Following, an example of use of the *CH_MONITOR* macro is provided:

```
#define SC_DEBUG
// this preprocesor command if SC_DEBUG not configured during the installation
#include "hetsc.h"
SC_MODULE(my_module) {
    sc_port<…>   out_port;
    uc_fifo<…>    *my_channel ;
    void process_functionality() {
       // computation …
       CH_MONITOR(out_port->write(a);)  // out_port
       // computation …
       CH_MONITOR(inner_ch->read(b);)
       // computation …
    }
   SC_CTOR(my_module) { …
          // declare process_functionality as SC_THREAD
          // and instances the inner channel my_channel   …
    }
};
```

1.10.4 Specification utilities for Profiling

A *BEGIN_PROF()* macro at the beginning of each process can be included. This is required when the Perfidy GIM/UC library is used to profile the HetSC specification. When the Perfidy library is used, the PERF macro is enables. This currently disables the utility for concurrent debugging. In the specification, the *CH_MONITOR* macro is used together with the *BEGIN_PROF()* for dividing the code into segments and obtain the system-level profile information. Following, and example of use of the *BEGIN_PROF()* macro is given:

```
void process_functionality() {
   BEGIN_PROF();
   …
   CH_MONITOR(out_port->write(a);)
   …
   CH_MONITOR(inner_ch->read(b);)
   …
}
```